GAUSS ELIMINATION

Textbook (Python) Part III Chapter 9

From now until the midterm exam

- **Week 7**
  - Solving systems of linear equations
- **Office Hour**
  - Friday Feb 23, 1:30 – 3:30, Lab203
- **Week 8**
  - Review and practice exercises
  - No office hour on Friday, March 1
  - Midterm exam the week after TBA

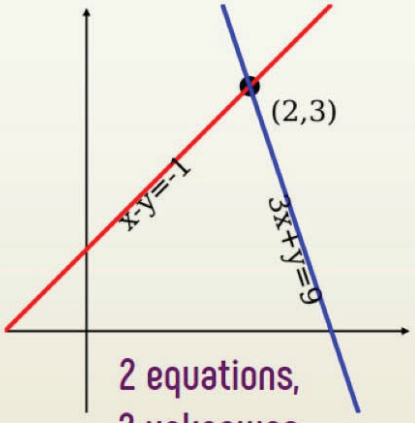
Systems of linear equations

System of Linear Equation

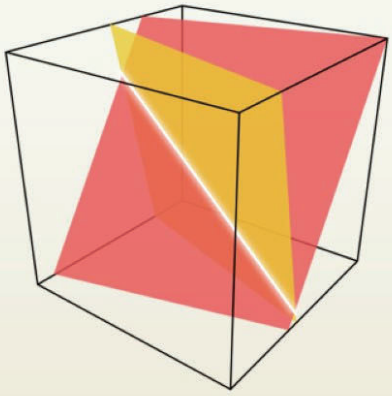
$$2.0x + 4.0y + 6.0z = 18$$
$$4.0x + 5.0y + 6.0z = 24$$
$$3.0x + 1y - 2.0z = 4$$

Matrix representation

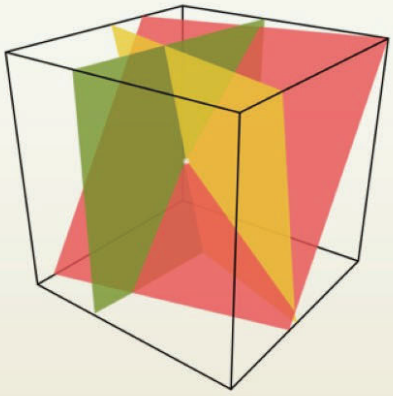
$$A = \begin{bmatrix} 2.0 & 4.0 & 6.0 \\ 4.0 & 5.0 & 6.0 \\ 3.0 & 1.0 & -2.0 \end{bmatrix} \quad X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad b = \begin{bmatrix} 18.0 \\ 24.0 \\ 4.0 \end{bmatrix}$$



2 equations,
2 unknowns

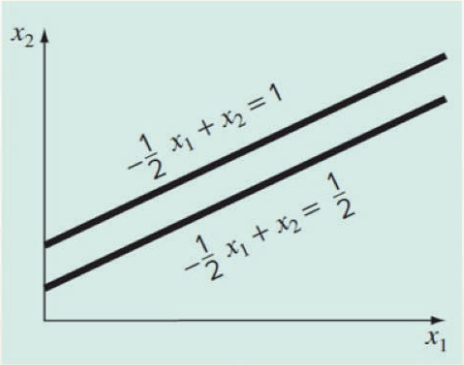


2 equations, 3 unknowns

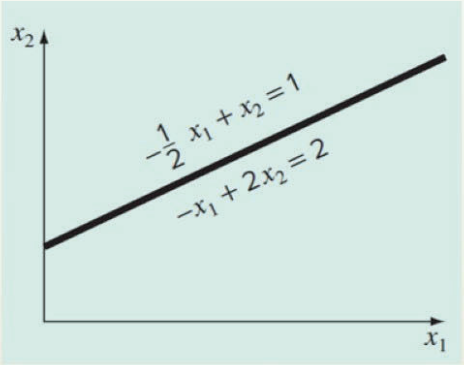


3 equations, 3 unknowns

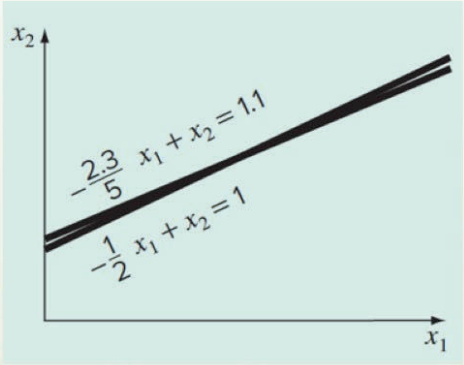
Singular and ill-condition systems



no solution



infinite solutions



ill-condition

Systems of linear equations

- Consider the following system of equations

$$\left.\begin{aligned} 4a - 2b + c &= 8 \\ a + b + c &= -13 \\ 9a + 3b + c &= 3 \end{aligned}\right\}$$ *Goal: find the values of a, b, and c that satisfy every equation in the system*

- Representing it as a matrix equation

$$\begin{bmatrix} 4 & -2 & 1 \\ 1 & 1 & 1 \\ 9 & 3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 8 \\ -13 \\ 3 \end{bmatrix}$$

Coefficient matrix | Variable vector | Constant vector

This can be solved using matrix inverse

$$\begin{aligned} \boldsymbol{Ax} &= \boldsymbol{b} \\ \boldsymbol{A^{-1}Ax} &= \boldsymbol{A^{-1}b} \\ \boldsymbol{Ix} &= \boldsymbol{A^{-1}b} \\ \boldsymbol{x} &= \boldsymbol{A^{-1}b} \end{aligned}$$

WORKED EXAMPLES

- Consider the following system of equations, solve it, i.e. find a, b, and c.

$$4a - 2b + c = 8$$

$$a + b + c = -13$$

$$9a + 3b + c = 3$$

WORKED EXAMPLES

- Consider the following system of matrix equation, solve it using inverse

$$\begin{bmatrix} 4 & -2 & 1 \\ 1 & 1 & 1 \\ 9 & 3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 8 \\ -13 \\ 3 \end{bmatrix}$$

Cramer's rule

- Best suited for small numbers of equations
- Consider a system of n linear equations for n unknowns represented in a matrix multiplication form:

$$\boldsymbol{Ax} = \boldsymbol{b}$$

- We can solve for x_i as:

$$x_i = \frac{\det(A_{i|b})}{\det(A)}$$

where $A_{i|b}$ is the matrix formed by replacing the i-th column of A by the constant vector b

WORKED EXAMPLES

- Use the Cramer's rule to solve the following system of equations:

$$\begin{bmatrix} 4 & -2 & 1 \\ 1 & 1 & 1 \\ 9 & 3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 8 \\ -13 \\ 3 \end{bmatrix}$$

Naïve Gauss Elimination

$$\left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array}\right]$$

$$\downarrow$$

$$\left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1 \\ & a'_{22} & a'_{23} & b'_2 \\ & & a''_{33} & b''_3 \end{array}\right]$$

(a) Forward elimination

$$\downarrow$$

$$x_3 = b''_3/a''_{33}$$

$$x_2 = (b'_2 - a'_{23}x_3)/a'_{22}$$

$$x_1 = (b_1 - a_{13}x_3 - a_{12}x_2)/a_{11}$$

(b) Back substitution

- **Forward elimination** of unknowns – reduce the set of equations (*in an augmented matrix representation*) to an upper triangular matrix
- **Back substitution** – back substitute into the $(n-1)$-th equation to solve for x_{n-1}, then work backward to x_{n-2} and down to x_1

Naïve Gauss Elimination

- First, represent the system of linear equations by the **augmented matrix**. Then, systematically apply **row operations** until we have the upper triangular matrix form:

$$[A|b] = \left[\begin{array}{ccc|c} 4 & -2 & 1 & 8 \\ 1 & 1 & 1 & -13 \\ 9 & 3 & 1 & 3 \end{array}\right]$$

Forward elimination of unknowns

Back substitution

- Working from bottom up (from the last row to the first)

Gauss-Jordan elimination

- Setting it up in another way so **the inverse of the coefficient matrix** can be obtained using the same forward Gauss elimination method and by systematically applying the back substitution

$$[A|I] = \left[\begin{array}{ccc|ccc} 4 & -2 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 9 & 3 & 1 & 0 & 0 & 1 \end{array}\right]$$

Forward elimination of unknowns

Backward elimination

LU FACTORIZATION

Textbook (Python) Part III Chapter 10

LU Factorization

- Gauss elimination is designed to solve systems of linear algebraic equations: $[A]\{x\} = \{b\}$
- It becomes inefficient when *solving equations with the same coefficients* $[A]$, *but with different right-hand-size constants* $\{b\}$

➔ LU **factorization** separates the time-consuming elimination of the matrix $[A]$ from the manipulations of the right-hand-side $\{b\}$. Once $[A]$ is decomposed, multiple right-hand-side vectors can be evaluated in an efficient manner.

LU Factorization

- $[A] = [L][U]$ decomposing A into *lower and upper triangular* matrices

$$L = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \qquad U = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

- $[A]\{x\} = \{b\}$

I. $[L]\{d\} = \{b\}$ solve for a vector d using *forward substitution*

II. $[U]\{x\} = \{d\}$ solve for a vector x using *backward substitution*

Gauss elimination to compute LU factorization

- The forward elimination step reduces $[A]$ to upper triangular matrix $[U]$
- The matrix [L] is produced also during the gauss elimination steps
  - l_{ij} is equal to the factor used at the step when A_{ij} is eliminated

$$A \begin{bmatrix} 4 & -2 & 1 \\ 1 & 1 & 1 \\ 9 & 3 & 1 \end{bmatrix} \qquad L \begin{bmatrix} 1 & 0 & 0 \\ 0.25 & 1 & 0 \\ 2.25 & 5 & 1 \end{bmatrix} \qquad U \begin{bmatrix} 4 & -2 & 1 \\ 0 & 1.5 & 0.75 \\ 0 & 0 & -5 \end{bmatrix}$$

Gauss elimination - forward elimination of unknowns

LU Factorization for solving systems of equations

Naïve Guass

- *This source code is from the textbook, fig9.4*


```
import numpy as np
def gaussnaive(A,b):
    """ gaussnaive: naive Gauss elimination
    input:
    A = coefficient matrix
    b = constant vector
    output:
    x = solution vector """
    (n,m) = A.shape
    if n != m:
       return 'Coefficient matrix A must be square'
    nb = n+1
    # build augmented matrix
    Aug = np.hstack((A,b))
    # forward elimination
    for k in range(n-1):
      for i in range(k+1,n):
        factor = Aug[i,k]/Aug[k,k]
        Aug[i,k:nb] = Aug[i,k:nb]-factor*Aug[k,k:nb]
    # back substitution
    x = np.zeros([n,1]) # create empty x array
    x = np.matrix(x) # convert to matrix type
    x[n-1] = Aug[n-1,nb-1]/Aug[n-1,n-1]
    for i in range(n-2,-1,-1):
        x[i] = (Aug[i,nb-1]-Aug[i,i+1:n]*x[i+1:n,0])/Aug[i,i]
    return x
```

```
A = np.matrix('4 -2 1 ; 1 1 1 ; 9 3 1', dtype=np.float64)
gaussnaive(A, np.matrix('8; -13; 3')
```


From the Naïve Gauss source code ...

- Write Gauss Jordan
  - You will need a backward elimination process
- Extend the Gauss Jordan to get LU factorization
- To solve $[A]\{x\} = \{b\}$ using LU factorization
  - You will need both forward and backward substitution processes
- Note: the `scipy` package contains the `linalg.lu(A)` function
  - You may get different results using this function. The differences are due to naïve vs. pivoting (not covering in this lecture) in the elimination process.